Mustererkennung - Aufgabenblatt 09

André Hacker und Dimitri Schachmann

2. AdaBoost

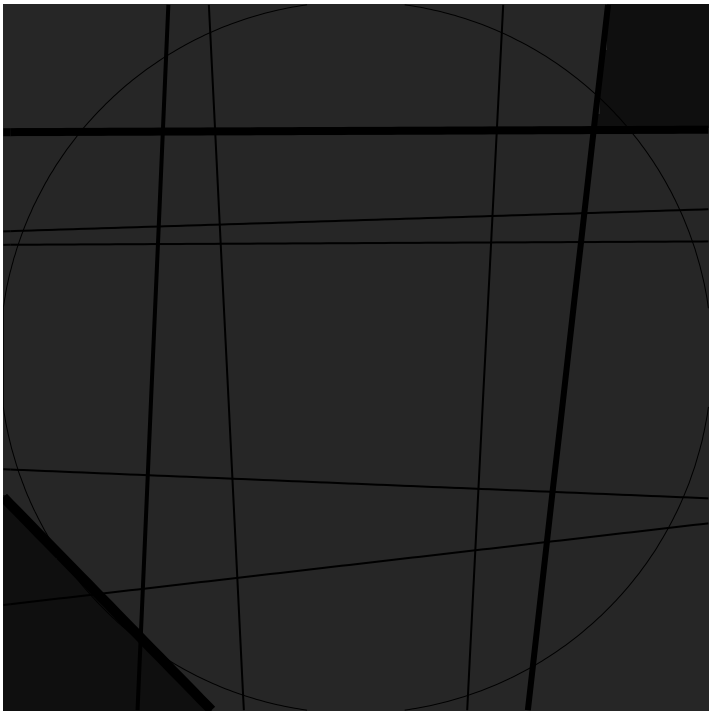
Wir haben für das Weihnachtsübungsblatt vor allem Aufgabe 2 bearbeitet und dafür eine Ausprägung des AdaBoost Algorithmus implementiert.

```
% Task 2 - recognizing a circle with lines and adaboost.
function adaboost
  fieldsize = 50;    % field will be twice as big
  squadsize = 100000; % number of randomy generated lines
  teamsize  = 10;    % number of classifiers picked by adaboost
  circle_proportion = 0.5;
  data = create_data(fieldsize, circle_proportion); % create the original circle data

  % randomly create lines in polar representation
  [squad_radius squad_theta] = create_classifiers(fieldsize, ...
  squadsize);

  % parameters of the dream team
  team_radius = zeros(teamsize,1);
  team_theta = zeros(teamsize,1);
  team_weight = zeros(teamsize,1);

  % stores the weights of the data points
  data_weight = ones(size(data));
  % weight for white pixels
  white_w = 1/sum(sum(data==1));
  % weight for black pixels
  black_w = 1/sum(sum(data==-1));
  data_weight(data==1) = white_w;
  data_weight(data==-1) = black_w;

  % stores the current weighted error
  weighted_error = ones(squadsize,1);

  % enumeration of all coordinates. just for utility.
  x_c = repmat([-fieldsize:fieldsize],fieldsize*2+1,1);
  y_c = repmat([-fieldsize:fieldsize]',1,fieldsize*2+1);

  % precompute the classification of all lines. No need to do it
  % every iteration
  scout_matrix = compute_scout_matrix(squad_radius, squad_theta, data, x_c, y_c);
  for m = [1:teamsize]
    [m teamsize]
    for l = [1:size(squad_radius,1)]
      % test all lines with current data weights
      weighted_error(l) = sum(sum(scout_matrix(:,:,l) .* data_weight));
    end
```

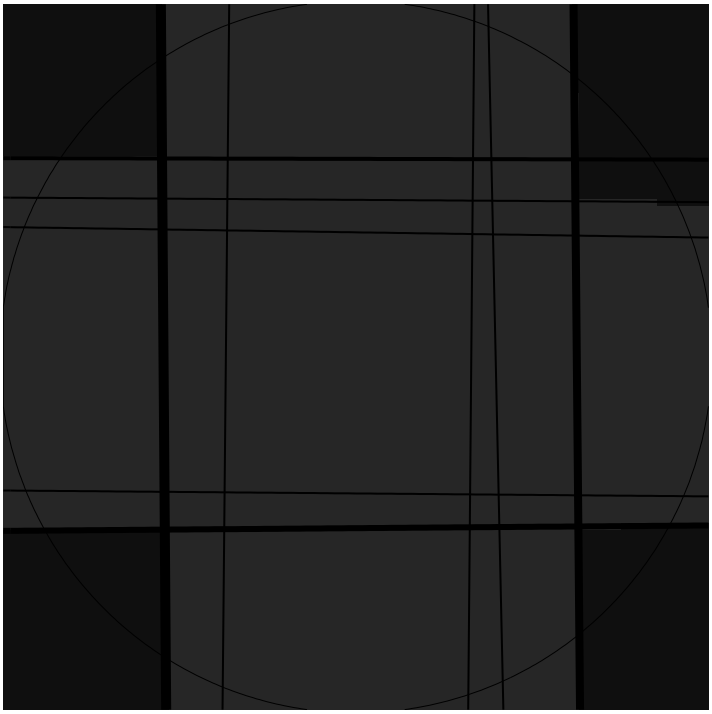
```
    % get the best classifiere
    [val index] = min(weighted_error);
    % compute the current sum of all data weights
    sum_of_weights = sum(sum(data_weight));

    % compute the error ratio
    e_m = (weighted_error(index)/sum_of_weights);

    % draw the best line into the dream team
    team_radius(m) = squad_radius(index);
    team_theta(m)  = squad_theta(index);

    % compute the weight for the classifiere
    alpha_m = (log((1-e_m)/e_m))/2;
    team_weight(m) = alpha_m;

    % update data weights
    data_weight = ((scout_matrix(:,:,index)==1) .* exp(alpha_m) ...
+ (scout_matrix(:,:,index)==0) .* exp(-0.5*alpha_m)) .* data_weight;
  end

  dream_team = [team_radius team_theta team_weight];

  % test the dream team
  c = classify(dream_team,fieldsize);


  %%%
  % Plotting
  %%%

  h = figure;
  % draw the predicted data
  imagesc([-fieldsize fieldsize],[-fieldsize fieldsize],sign(c));
  axis equal off;

  % draw the picked lines
  [not_used sl] = sort(team_weight,'descend');
  lw = 5; % thikness of line
  for i = [1:min(size(dream_team),50)]
    hold on;
    x = -fieldsize:fieldsize;
    y = (team_radius(sl(i)) + x* cos(team_theta(sl(i))) )/sin(team_theta(sl(i)));
    p = plot(x,y);

    set(p,'Color','green','LineWidth',lw);
    lw = max(lw - 1, 1);
    ylim([-fieldsize fieldsize]);
    xlim([-fieldsize fieldsize]);
  end

  % draw original circle for reference
  t=0:.01:2*pi;
```

```
  plot(circle_proportion*size(data,1)*sin(t),circle_proportion*size(data,1)*cos(t));
  print(h,'-deps',['task2-original.eps']);
  hold off;
end

% lets a team of lines classify a circle
function result = classify(team,fieldsize)
  x = repmat([-fieldsize:fieldsize],fieldsize*2+1,1);
  y = repmat([-fieldsize:fieldsize]',1,fieldsize*2+1);
  result = 0;
  for i = [1:min(size(team,1))]
    r = team(i,1);
    t = team(i,2);
    w = team(i,3);
    prediction = (sign(-x*cos(t)+y*sin(t)-r)*-1);
    result = result + prediction*w;
  end
  result = single(result >= 0.2*sum(team(:,3)));
end


% compute the scout matrix for a set of lines
% given by r(adius) and t(heta)
% d is the data
% x and y are 2D-lists of all coordinates
function S = compute_scout_matrix(r, t, d, x, y)
  S = zeros([size(d) size(r)]);
  for i = [1:size(r)]
    S(:,:,i) = ((sign(-x*cos(t(i))+y*sin(t(i))-r(i))*-1)~=d);
  end
end

% create a circle image (by Barbara Haupt)
function data = create_data(fieldsize, circle_proportion)
  [xs,ys]=meshgrid(-fieldsize:fieldsize);
  data=ones(size(xs));
  data(sqrt(xs.^2+ys.^2)>=(circle_proportion*size(xs,1)))=-1;
end

% create random classifiers
function [radius angle] = create_classifiers(fieldsize,number)
  max_radius = sqrt((fieldsize*2+1)^2 + (2*fieldsize+1)^2)/2;
  radius = rand(number,1)*max_radius;
  angle  = rand(number,1)*360;
end
```

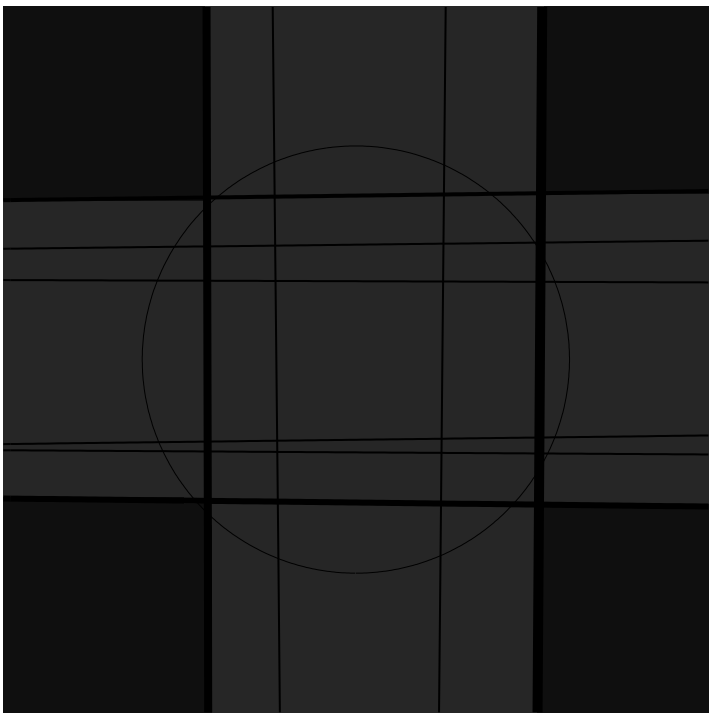
Bei der Auswertung haben wir festgestellt, dass wir ein anderes Ergebnis bekommen haben, als erwartet. Wir konnten nicht beobachten, dass der Algorithmus zuverlässig Tangenten zum Kreis auswählt. Dies ist nämlich stark von dem Verhältnis des Kreisradius zur Bildgröße abhängig. So konnten wir allgemein nur dann Tangenten beobachten, wenn der Kreis an den Bildrand gestoßen ist:



Dieses Bild kommt aber auch nur zu Stande, wenn wir die Datengewichte alle gleich initialisieren. Wenn wir sie aber so wie vorgeschlagen initialisieren (siehe Quellcode), dann ist es ja "nicht so schlimm" wenn weiße Pixel falsch klassifiziert werden. Dann bekommen wir das folgende Bild:

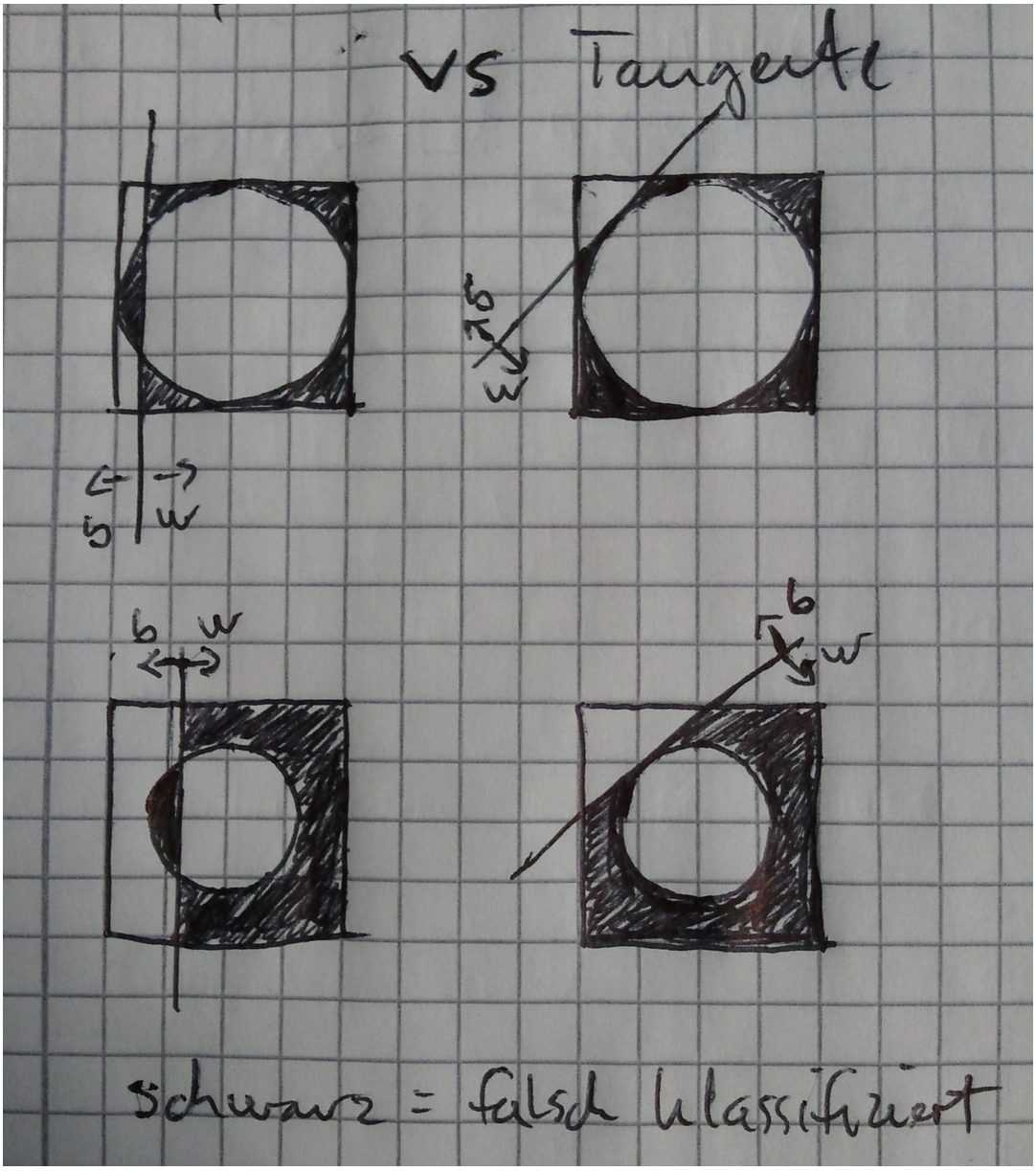


Falls der Kreis sehr klein ist, dann verlaufen die Linien durch den Kreis. Das kann man hier gut beobachten:



Das ergibt für uns Sinn, wenn wir uns die Fehlerfunktion und den Algorithmus ansehen. Insbesondere kann man das bei dem folgenden Bild nachvollziehen. Die vertikalen und horizontalen Linien, die den Kreis schneiden klassifizieren deutlich weniger Fläche falsch, als jede vorstellbare Tangente. Die Gewichte der Daten werden zwar ständig angepasst, aber bis es vermeintlich endlich so weit ist, hat der Algorithmus sehr viele schädliche Linien ausgewählt. Ansonsten konnten wir überhaupt nicht beobachten, dass überhaupt irgendwann Tangenten bei einem kleinen Kreis ausgewählt wurden. Entscheidend ist, dass AdaBoost ein greedy Algorithmus ist.

AdaBoost ist ein greedy Algorithmus. Um ihn zu verstehen, ist es wichtig den ersten Schritt anzusehen, also die Wahl des ersten Klassifikators. In der folgenden Darstellung haben wir für einen großen und einen kleineren Kleis illustriert, warum die Tangente manchmal nicht als erstes gewählt wird. Die falsch klassifizierten Bereiche sind schwarz.
Beim großen Kreis (oben) ist nicht klar, ob die Wahl der Tangente besser ist als die Wahl der Gerade durch den Kreis, da die Unterschiede nur gering sind. Nur in diesem Fall haben wir Tangenten beobachtet.
Beim kleineren Kreis (unten) wird sichtbar, dass die Wahl der Gerade durch den Kreis die beste Wahl für den ersten Klassifikator ist. Im linken Bild sind die falsch klassifizierten schwarzen Flächen geringer als im rechten (mit der Tangente).
Die späteren Schritte sind sehr komplex nachzuvollziehen (aufgrund der geänderten Gewichte) und es ist nicht trivial eine Argumentation zu finden, die die Wahl von Tangenten erklären würde.


vs Tangente
b
w
b
w
b
w
schwarz = falsch klassifiziert